REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

AD-A238 074

1. AGE

DATE

3. REPORT TYPE AND DATES COVERED
Final: 18 Apr 1991 to 01 Mar 1993

4. TITLE AND SUBTITLE
IBM Canada, Ltd., AIX Ada/6000 Release 2, Preliminary Version, RISC System/600 model 7013-530 (Host & Target), 901127W1.11085

5. FUNDING NUMBERS

6. AUTHOR(S)
Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB
Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF-VSR-438.0491

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
IBM Canada, Ltd., AIX Ada/6000 Release 2, Preliminary Version, Wright-Patterson AFB, OH, RISC System/600, model 7013-530, (Host & Target), ACVC 1.11.

91-03870

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 27 November 1990.

Compiler Name and Version: AIX Ada/6000 Release 2, Preliminary Version

Host Computer System: RISC System/6000, model 7013-530 under AIX 3.1

Target Computer System: RISC System/6000, model 7013-530 under AIX 3.1

See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 901127W1.11085 is awarded to IBM Canada, Ltd. This certificate expires on 1 March 1993.

This report has been reviewed and is approved.

Steven E. Wilson

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

John P. Solomond

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

AVF Control Number: AVF-VSR-438.0491
18 April 1991
90-09-13-IBM

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 901127W1.11085
IBM Canada, Ltd.
AIX Ada/6000 Release 2, Preliminary Version
RISC System/6000, model 7013-530 => RISC System/6000, model 7013-530

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright Patterson AFB OH 45433-6503

DECLARATION OF CONFORMANCE
========== == ==========

Customer: IBM Canada Ltd.

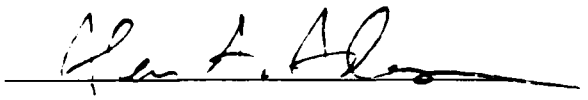
Ada Validation Facility: Wright Patterson AVF
ASD/SCEL
WPAFB OH 45433
USA

ACVC Version: 1.11

Ada Implementation:
Compiler Name and Version: AIX Ada/6000 Release 2, Preliminary Version
Host Computer System: RISC System/6000, model 7013-530, AIX 3.1
Target Computer System: RISC System/6000, model 7013-530, AIX 3.1

Customer's Declaration

I, the undersigned, representing IBM Canada Ltd., declare that IBM Canada Ltd. has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.



Date: Nov 27/1990

Alan A. Adamson
IBM Canada Ltd. Laboratory
844 Don Mills Road
North York, Ontario
CANADA M3C 1V7

CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.

Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 21 November 1990.

E28005C	B28006C	C34006D	C35702A	B41308B	C43004A
C45114A	C45346A	C45612B	C45651A	C46022A	B49008A
A74006A	C74308A	B83022B	B83022H	B83025B	B83025D
B83026B	B85001L	C83026A	C83041A	C97116A	C98003B
BA2011A	CB7001A	CB7001B	CB7004A	CC1223A	BC1226A
CC1226B	BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A
CD2A21E	CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A
CD2B15C	BD3006A	BD4008A	CD4022A	CD4022D	CD4024B
CD4024C	CD4024D	CD4031A	CD4051D	CD5111A	CD7004C
ED7005D	CD7005E	AD7006A	CD7006E	AD7201A	AD7201E
CD7204B	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

IMPLEMENTATION DEPENDENCIES

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

C35508I..J and C35508M..N (4 tests) include enumeration representation clauses for boolean types in which the specified values are other than (FALSE => 0, TRUE => 1); this implementation does not support a change in representation for boolean types. (See section 2.3.)

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 48 or greater.

C45624A checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 5. For this implementation, MACHINE_OVERFLOWS is TRUE.

C45624B checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 6. For this implementation, MACHINE_OVERFLOWS is TRUE.

C86001F recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete. For this implementation, the package TEXT_IO is dependent upon package SYSTEM.

B86001Y checks for a predefined fixed-point type other than DURATION.

CA2009C and CA2009F instantiate generic units before the generic bodies are compiled. (See section 2.3).

LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F check for pragma INLINE for procedures and functions.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A840 use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The tests listed in the following table are not applicable because the given file operations are supported for the given combination of mode and file access method:

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	--------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

EE2401D uses an instantiation of package DIRECT_IO with unconstrained array types. An attempt to create a file of this type raises USE_ERROR at run-time since the size of the file exceeds the capacity of this implementation.

IMPLEMENTATION DEPENDENCIES

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE2107B..E	CE2107G..H	CE2107L	CD2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3304A checks that USE_ERROR is raised if a call to SET_LINE_LENGTH or SET_PAGE_LENGTH specifies a value that is inappropriate for the external file. This implementation does not have inappropriate values for either line length or page length.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST. For this implementation, the value of COUNT'LAST is greater than 150000 making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 26 tests:

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

BA1001A	BA2001C	BA2001E	BA3006A	BA3006B
BA3007B	BA3008A	BA3008B	BA3013A	

C35508I..J and C35508M..N (4 tests) were graded inapplicable by Evaluation Modification as directed by the AVO. These tests attempt to change the representation of a boolean type. The AVO ruled that, in consideration of the particular nature of boolean types and the operations that are defined for the type and for arrays of the type, a change of representation need not be supported; the ARG will address this issue in Commentary AI-00564.

C52008B was graded passed by Test Modification as directed by the AVO. This test uses a record type with discriminants with defaults and that has array components whose size depend on the values of some discriminants of type INTEGER. On compilation of the type declaration, this implementation

raises NUMERIC_ERROR as it attempts to calculate the maximum possible size for objects of the type. Although this behavior is a violation of the Ada Standard, the AVO ruled that the implementation be accepted for validation in consideration of intended changes to the standard to allow for compile-time detection of run-time error conditions. The test was modified to constrain the subtype of the discriminants. Line 16 was modified to declare a constrained subtype of INTEGER, and discriminant declarations in lines 17 and 25 were modified to use that subtype; the lines are given below:

```
16  SUBTYPE SUBINT IS INTEGER RANGE -128 .. 127;
17  TYPE REC1(D1,D2 : SUBINT) IS

25  TYPE REC2(D1,D2,D3,D4 : SUBINT := 0) IS
```

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete--no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected.

CD1009A, CD1009I, CD1C03A, CD2A21C, CD2A24A, and CD2A31A..C (3 tests) were graded passed by Evaluation Modification as directed by the AVO. These tests use instantiations of the support procedure LENGTH_CHECK, which uses UNCHECKED_CONVERSION according to the interpretation given in AI-00590. The AVO ruled that this interpretation is not binding under ACVC 1.11; the tests are ruled to be passed if they produce Failed messages only from the instantiations of LENGTH_CHECK--i.e., the allowed Report.Failed messages have the general form:

" * CHECK ON REPRESENTATION FOR <TYPE_ID> FAILED."

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada implementation system, see:

IBM Canada, Ltd
844 Don Mills Road
North York, Ontario
Canada M3C IV7
ATTN: Graham Peace
21/199/844/TOR

For a point of contact for sales information about this Ada implementation system, see:

IBM Canada, Ltd
844 Don Mills Road
North York, Ontario
Canada M3C IV7
ATTN: Bob Gerber
21/634/844/TOR

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

PROCESSING INFORMATION

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

a) Total Number of Applicable Tests	3777
b) Total Number of Withdrawn Tests	83
c) Processed Inapplicable Tests	109
d) Non-Processed I/O Tests	0
e) Non-Processed Floating-Point Precision Tests	201
f) Total Number of Inapplicable Tests	310
g) Total Number of Tests for ACVC 1.11	4170

All I/O tests of the test suite were processed because this implementation supports a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 310 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the tape were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing when appropriate during this test were:

Option	Effect
-l	Produce a listing file containing the lines from the Ada source code interspersed with any errors the compiler finds.
-b UnitName	Produce an executable file using UnitName as the main program unit.
-m	Compile a source file and produce an executable file. The main unit of the program is the last compilation unit in the source file. NOTE: This option is mutually exclusive with the -b option.
-u	Unlock the working sublibrary so that the compiler can access it and update it.

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_IN_LEN	200
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

MACRO PARAMETERS

$MAX_STRING_LITERAL '"' & (1..V-2 => 'A') & '"'

The following table lists all of the other macro parameters and their respective values:

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_646
$DEFAULT_MEM_SIZE	268_435_456
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	AIX_6000
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	ENTRY0'ADDRESS
$ENTRY_ADDRESS1	ENTRY1'ADDRESS
$ENTRY_ADDRESS2	ENTRY2'ADDRESS
$FIELD_LAST	1000
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100_000.0
$GREATER_THAN_DURATION_BASE_LAST	131_073.0
$GREATER_THAN_FLOAT_BASE_LAST	1.80141E+38
$GREATER_THAN_FLOAT_SAFE_LARGE	1.0E308

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        1.0E308

$HIGH_PRIORITY          255

$ILLEGAL_EXTERNAL_FILE_NAME1
                        A*

$ILLEGAL_EXTERNAL_FILE_NAME2
                        *B

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        'PRAGMA INCLUDE ("A28006D1.TST");'

$INCLUDE_PRAGMA2        'PRAGMA INCLUDE ("B28006F1.TST");'

$INTEGER_FIRST          -2147483648

$INTEGER_LAST           2147483647

$INTEGER_LAST_PLUS_1    2147483648

$INTERFACE_LANGUAGE     FORTRAN

$LESS_THAN_DURATION     -100_000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -131_073.0

$LINE_TERMINATOR        ASCII.LF

$LOW_PRIORITY           0

$MACHINE_CODE_STATEMENT
                        NULL;

$MACHINE_CODE_TYPE      NO_SUCH_TYPE

$MANTISSA_DOC           31

$MAX_DIGITS             15

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -2147483648
```

MACRO PARAMETERS

$NAME	NO_SUCH_TYPE_AVAILABLE
$NAME_LIST	AIX_6000
$NAME_SPECIFICATION1	/u/acvc111/new/run/tmp/ctests/ce/X2120A
$NAME_SPECIFICATION2	/u/acvc111/new/run/tmp/ctests/ce/X2120B
$NAME_SPECIFICATION3	/u/acvc111/new/run/tmp/ctests/ce/X3119A
$NEG_BASED_INT	16#F000000E#
$NEW_MEM_SIZE	65535
$NEW_STOR_UNIT	16
$NEW_SYS_NAME	AIX_6000
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	"NEW INTEGER;"
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	8192
$TICK	0.00006
$VARIABLE_ADDRESS	VARIABLE'ADDRESS
$VARIABLE_ADDRESS1	VARIABLE1'ADDRESS
$VARIABLE_ADDRESS2	VARIABLE2'ADDRESS
$YOUR_PRAGMA	EXPORT_OBJECT

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report.

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to linker documentation and not to this report.

SUMMARY OF COMPILER OPTIONS

The following figures list compiler options by function. From left to right, the columns show:

1. The syntax you use to specify the option on a command line. Some options require you to enter another parameter separated by a blank. For example, the -b option requires that you also specify an Ada compilation unit:

   ada -b UnitName FileName

2. A short description of the option's function.
3. The default. This is what happens if the option is not specified on the command line.

NOTE: Before using the AIX Ada/6000 compiler, it is important that the programmer first read the descriptions of the compiler options to understand the correct operation and limitations of this product.

Table 1-1 Options for Compiler Output

COMPILER OPTION	DESCRIPTION	DEFAULT
-v	Display all messages.	Compile without displaying a copyright notice or progress messages.
-o Name	Name the executable or object output file Name.	The name of the executable or object output file is a.out.
-l	Produce a listing file containing the lines from the Ada source files interspersed with any errors the compiler finds. The listing file has a suffix of ".lst".	Display error listings on screen only.
-a	Produce an assembler code listing. The assembler output file has a suffix of ".s".	Do not generate an assembler listing.
-G	Produce optimized code. The level of optimization does not prevent you from debugging this code, but some source information may be removed or changed during compilation.	Do not perform code optimization.
-O	Produce highly optimized code. The level of optimization prevents the debugger from working with this code because much of the organization present in the source is changed. The optimizations include all those done by the -G option, so do not specify both options.	Do not perform code optimization.
-p	Produce profiled code. Specify this option for each compilation operation on a compilation unit, whether or not an executable file is generated.	Do not produce profiled code.
-Q	If all single-precision floating-point overflows must be detected, use this option.	

Table 1-2 Options for Linking

COMPILER OPTION	DESCRIPTION	DEFAULT
-b UnitName	Produce an executable file using UnitName as the main program unit. You can omit the Ada source file name if you want to produce an executable file from compilation units that have all been compiled.	Do not compile all the way to an executable file.
-m	Compile a source file and produce an executable file. The main unit of the program is the last compilation unit in the source file.	Do not produce an executable file.
-e	Produce an object file if -b or -m is specified. The file has a default name of a.out and is put in your current directory.	Produce an executable file when -b or -m is specified on the command line.
-i FileName	During linking, include an object file containing subprograms written in other languages. The object file should have a suffix of ".o" or ".a".	The executable output file contains only modules written in Ada.

Table 1-3. Options for Libraries

COMPILER OPTION	DESCRIPTION	DEFAULT
-L LibraryList	Use the file LibraryList as the library list file.	The default library list name is alib.list.
-u	Unlock the working sublibrary so that the compiler can access and update it.	Stop compiling if another compilation stopped without finishing and left the sublibrary inaccessible.
-d	Store information for debugging in the working sublibrary.	Do not store information for debugging in the working sublibrary.

Table 1-4 Options for Other Compiler Features

COMPILER OPTION	DESCRIPTION	DEFAULT
-I	Read a list of file names from standard input and compile them one after another. Do not update the working sublibrary if any compilation errors occur in any file.	Name of a single file to compile is specified on the command line.
-S	Suppress checks on all data types as if pragma SUPPRESS was applied.	Do full checking.
-t	Use the special memory layout required to support pragma OS_TASK.	Use the memory layout of the default tasking facility.
-V Number	Specify the number of pages for the compiler to use in managing memory storage.	3000 pages, 1K bytes each.

DETAILED DESCRIPTION OF COMPILER OPTIONS

Following are more detailed descriptions of the compiler options. The descriptions include information about how the options interact with each other, and any restrictions on their use.

Options for Compiler Output

-v Print the compiler banner and version notice, then display progress messages during compilation. By default no notices or progress messages are displayed.

-o Name Specify the name for the executable output file. If you do not use this option, the file name is a.out.

Because this option only has an effect when you generate an executable or object file, do not specify it unless you also specify the -b or -m options.

-l Create a file (with the suffix ".lst") containing a source listing and a list of all syntax and semantic errors the compiler finds in the Ada source file. The Ada source lines are preceded by line numbers to make it easier to locate specific places within the source file. Any errors the compiler finds are printed below the line where the error was found.

The listing file also contains the compiler banner message and compilation statistics.

-a Write assembler code to an output file that has a suffix of ".s". Note that the resulting file contains a concatenation of all assembler code produced together with information to map source lines to assembler statements, and so may not be in a legal format for the AIX assembler. This occurs most often in the case of main unit compilations, where the main elaboration code and the main program code are concatenated into the same file.

The generated assembler code is most useful in locating problems, or in verifying how the compiler is optimizing certain constructs.

-G Perform a moderate level of code optimization. Specify the -G option to make the compiler analyze the source as it compiles and generate code that is faster and possibly smaller. The compiler takes longer to process each compilation unit when you specify this option.

The debugger can work with code compiled at this level of optimization, but information that comes from Ada source files (such as which source line corresponds to a set of machine instructions) may not be entirely accurate due to reorganization done during optimization.

-O Perform extensive code optimizations, including inline expansion. The optimizations done by the -G option are done at a higher level by the -O option. If you use -O there is no need to specify -G.

Code compiled with the -O option does not work with the debugger at all because the structure of the generated code differs substantially from that of the Ada source.

-p Produce code that records profiling information by calling system profiling routines. To view the profile data:

Run a program compiled with this option; this produces a file called "gmon.out".
Use the AIX Ada/6000 aprof command to produce a profile report.

NOTES:

1. The compiler performs special operations in each phase of compilation when you compile with profiling. Thus if you compile a unit without generating an executable file, then later compile with the -b or -m options to produce an executable file, specify the -p option each time you run the compiler.

2. Your final program can contain a mixture of profiled and unprofiled compilation units. For units compiled without the -p option, the profile report only contains partial information.

3. Profiled units may run more slowly that unprofiled units. This is true even if you do not specify -p for the final compilation phases along with -b or -m.

-Q Generate floating-point no-op instructions to cause detection of overflow in rounding floating-point intermediate results to single precision.

Options for Linking

-b UnitName
Produce an executable file from previously compiled code. The UnitName parameter tells the compiler which compilation unit is the main unit.

With this option you can omit the Ada source file name at the end of the command line. Omitting the name of the source file causes the compiler to invoke the linkage editor to produce an executable file; all compilation units must already be compiled. If you do specify an Ada source file, that file will be compiled first, then UnitName will be linked as the main unit.

UnitName can be any compilation unit in the working sublibrary. It does not have to be part of the source file when you specify both a unit name and a source file name.

-m Compile the Ada source file specified at the end of the command line, and bind the last unit in that file as a main program.

NOTE: This option is mutually exclusive with the -I and -b options.

-e Permit the link step to complete successfully without necessarily resolving all symbols. Instead generate an object file. The file is named a.out unless you also specify the -o Name option, in which case the object file is Name. This file has all Ada references resolved, so that the only unresolved references are to subprograms specified through pragma INTERFACE.

Use this option only if you are very familiar with linking operations under the AIX operating system. You can use it to create an object file that can be linked with several different versions of non-Ada object modules, such as during development and testing of non-Ada interface packages.

NOTE: When linking the object file to produce an executable file, include the file "libada.a" in the link step. This file is in the AIX system directory named "/lib".

This option is valid only when the -m or -b options are also specified.

-i FileName
Include the object archive file named by FileName in the linking of the main program. Use this option to specify any non-Ada object code files that you need to link with your Ada program.

The -i option is ignored if neither -m nor -b was specified for the compilation. To specify multiple files to be linked with the Ada program, use -i once for each one, for instance:

```
ada -m -i assembler_functions.o -i c_archive.a ada_main_program.ada
```

Only use the -i option with files that have ".o" or ".a" suffixes, and specify the full name (including the suffix).

Options for Libraries

-L LibraryList

Specify the name for the library list file. The list of sublibraries in the library list file determines where the compiler searches for compilation units and dependency information, and the order in which it searches sublibraries.

If this option is not given, the compiler defaults to a library list named alib.list.

NOTE: The library name "liblst.tmp" is reserved by the compiler. Do not use this name for your own files.

-u Unlock the working sublibrary. Use this option to allow the compiler to use the working sublibrary again after a compilation has ended prematurely. For example you can use the AIX "kill" command to stop the compiler while it is updating the working sublibrary; then the compiler will not use that sublibrary again until you have run the compiler with the -u option.

As part of the unlocking, the compiler verifies that the sublibrary is not corrupted. Use the alibinit command to initialize any corrupted sublibraries.

You can specify other options along with -u; the compiler unlocks the sublibrary, then acts on any other specified options.

-d Store symbolic debugging information in the working sublibrary. This information makes it possible for the debugger to associate abstractions like line numbers and variable names with locations within a running program. Without it you cannot use the symbolic debugger with your program.

NOTES:

1. The debugging information can make the sublibrary very large.

2. The information produced by this option consumes more space in your working sublibrary and may cause your executable files to be larger; it does not cause executable files to require more run-time storage.

Options for Other Compiler Features

-I Direct the compiler to read a list of file names from standard input instead of taking a file name as a command-line parameter. Note the following restrictions:

- o If you do not pipe or redirect standard input, the compiler will pause and wait for you to type in the names of the files. Press Enter after each file name, and type the end-of-file character (usually Control-D) when you have entered all the file names.
- o This option is mutually exclusive with the -m, -b, and -e options. It only performs the initial compilation phase on each source file, not the later linking phase that these other options control.
- o If any file cannot be successfully compiled, the working sublibrary is not changed at all. The compiler stops processing files as soon as one compilation is unsuccessful.

You can put comments in the lines the compiler reads from standard input by using Ada comment syntax. The compiler ignores text on a line from a double hyphen (--) to the end of that line.

-S Suppress run-time checks on all types, as if pragma SUPPRESS was applied. This means that when error conditions such as constraint violation or numeric overflow occur at run time, the Ada program will not detect them and so will not raise the appropriate exceptions. The program may run faster because the checking is not performed; however you must take special care that no error conditions occur, otherwise your program will continue to run but may produce incorrect results.

-t Configure the memory layout of an executable program so that the special requirements of pragma OS_TASK are met. This option makes it possible for Ada tasks within a single program to run as separate AIX processes.

Only use this option to compile programs that apply pragma OS_TASK to some or all of their tasks. Specify it when you generate an executable program or object file, that is when you also specify the -b, -m, or -e options.

-V NumPages
Specify the number of virtual pages used by the compiler's virtual space manager. A larger number allows faster compilation speed, but demands more memory from the host system. In general, decrease NumPages if the compiler runs out of memory (raises a STORAGE_ERROR exception), and increase NumPages for systems with an abundance of memory for page space. The default is 3000. Each page holds 1K bytes.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...
   type INTEGER is range -2_147_483_648 .. 2_147_483_647;
   type SHORT_INTEGER is range -32_768 .. 32_767;

   type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;
   type LONG_FLOAT is digits 15 range -1.79769313486232E+308
                                   .. 1.79769313486232E+308;

   type DURATION is delta 2**(-14) range -86400.0 .. 86400.0;
   ...

end STANDARD;
```

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

The Ada language definition allows for certain target dependencies in a controlled manner. This appendix, called Appendix F as prescribed in the Ada Language Reference Manual, describes implementation-dependent characteristics of the AIX Ada/6000 compiler running under the AIX operating system Version 3.

IMPLEMENTATION-DEFINED PRAGMAS

Implementation dependent pragmas are:

```
PRAGMA COMMENT(String_Literal);
-- Imbeds String_Literal into object code.

PRAGMA IMAGES(Enumeration_Type, immediate | deferred);
-- Generates a table of images for the enumeration type.
-- "deferred" causes the table to be generated only if
-- the enumeration type is used in a compilation unit.

PRAGMA LINKNAME(Interfaced_Subprogram_Name, Link_Name);
-- When used in conjunction with pragma INTERFACE, provides
-- access to any routine whose name can be specified by an
-- Ada string literal.

PRAGMA OS_TASK (Priority);
-- Appears within the declaration for a task or task type
-- (in the same context as pragma PRIORITY),
-- and causes the task or task type to be placed into
-- a separate AIX process.
-- The priority value is of type SYSTEM.PRIORITY, and is not
-- currently acted upon.  To maintain upward compatibility,
-- always use a 0 for this parameter.
```

PREDEFINED PRAGMAS

Supported pragmas are INTERFACE, ELABORATE, SUPPRESS, PACK, PAGE, LIST, INLINE, and PRIORITY.

All pragmas have conventional meanings except LIST, which suppresses listings prior to pragma LIST(ON) regardless of the user request. Pragma INTERFACE supports C, FORTRAN, and assembler.

Unrecognized and unsupported pragmas are ignored with the appropriate warning message.

REPRESENTATION CLAUSES

Supported representation clauses include:

- Length Clause
- Enumeration Representation Clauses, except
  for boolean types
- Record Representation Clause
- Address Clause- Interrupt support

Records are aligned by default on 32-bit boundaries. You can use a representation clause to force them to be aligned on 64-bit boundaries.

RESTRICTIONS ON UNCHECKED CONVERSION

The only restriction on unchecked conversion is that the two types (or subtypes) A and B must be the same static size, and that neither A nor B are private.

PACKAGE SYSTEM

The Package System has the following characteristics:

```
package System is

-- for integer use 32;

   type Memory is private;
   type Address is access Memory;

   Null_Address : constant Address := null;

TYPE name IS (AIX_6000);

System_Name  : CONSTANT name := AIX_6000;

Storage_Unit : CONSTANT := 8;

Memory_Size  : CONSTANT := 1024*1024*256;
-- 256 Mb.

-- System-Dependent Named Numbers:

Min_Int      : CONSTANT :=  -(2**31);
Max_Int      : CONSTANT :=  (2 ** 31) -1;
Max_Digits   : CONSTANT :=  15;
Max_Mantissa : CONSTANT :=  31;
Fine_Delta   : CONSTANT :=  1.0 / (2 ** MAX_MANTISSA);
Tick         : CONSTANT :=  0.00006;

-- Other System-Dependent Declarations

SUBTYPE Priority IS integer RANGE 0..255;


END System;
```

REPRESENTATION ATTRIBUTES

All defined representation attributes shall be supported.

IMPLEMENTATION-GENERATED NAMES

There are no implementation-generated names denoting implementation-dependent components. Component names generated by the compiler shall not interfere with programmer-defined names.

IMPLEMENTATION-DEPENDENT CHARACTERISTICS OF THE I/O PACKAGES

- Packages SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO are supported.
- Package LOW_LEVEL_IO is not supported.
- Unconstrained array types and unconstrained types with discriminants may be instantiated for I/O.
- File names follow the conventions and restrictions of the target operating system, except that non-printing characters, blank(' ') and asterisk ('*') are disallowed.
- In TEXT_IO, the type Field is defined as follows: subtype Field is integer range 0..1000;
- In TEXT_IO, the type Count is defined as follows: type Count is range 0..16_384;

FORM PARAMETERS FOR FILE OPERATIONS

Section 14.2 of the Language Reference Manual describes the Ada functions for manipulating files. As stated in that section, the form string parameter allows you to set file protections when you create a file. The details of file protections and privileges under the AIX operating system are described under the "chmod" call in the AIX Calls and Subroutines Reference for IBM RISC System/6000 (SC23-2198).

If you do not specify a form string, the default file protection is both read and write privileges for the owner, group, and all others. If you do specify a form string, it is interpreted in the following way:

- The form string consists of a series of substrings, separated by blanks.
- The order of the substrings does not matter.
- Some substrings control the file protection settings. These substrings are case sensitive.
- Some substrings enable special AIX behavior for file opening, for example opening with no delay and opening a file for append. These substrings are not case sensitive.
- The list of recognized substrings is contained in the section entitled "Input/Output" in the User's Guide for IBM AIX Ada/6000 (SC09-1321).

PREDEFINED NUMERIC TYPES

The current specification of package STANDARD includes:

```
    type SHORT_INTEGER is range -32768 .. 32767;

    type INTEGER is range -2147483648 .. 2147483647;

    type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

    type LONG_FLOAT is digits 15 range -1.79769313486232E+308 ..
    1.79769313486232E+308;

    type DURATION is delta 2**(-14) range -86400.0 .. 86400.0;

    SHORT_INTEGER
        'First    = -32768
        'Last     =  32767
        'Size     =  16
    INTEGER
        'First    = -2147483648
        'Last     =  2147483647
        'Size     =  32
    FLOAT
        'Machine_Overflows    = TRUE
        'Machine_Rounds       = TRUE
        'Machine_Radix        = 2
        'Machine_Mantissa     = 24
        'Machine_Emax         = 128
        'Machine_Emin         = -125
        'Mantissa             = 21
        'Digits               = 6
        'Size                 = 32
        'Emax                 = 84
        'Safe_Emax            = 125
        'Epsilon              = 9.53674E-07
        'Safe_Large           = 4.25353E+37
        'Safe_Small           = 1.17549E-38
        'Small                = 2.58494E-26
        'Large                = 1.934280389046203E+25
```

```
    LONG_FLOAT
        'Machine_Overflows    = TRUE
        'Machine_Rounds       = TRUE
        'Machine_Radix        = 2
        'Machine_Mantissa     = 53
        'Machine_Emax         = 1024
        'Machine_Emin         = -1021
        'Mantissa             = 51
        'Digits               = 15
        'Size                 = 64
        'Emax                 = 204
        'Safe_Emax            = 1020
        'Epsilon              = 8.88178419700125E-16
        'Safe_Large           = 2.24711641857789E+307
        'Safe_Small           = 2.22507385850720E-308
        'Small                = 1.94469227433161E-62
        'Large                = 2.5711008708143832990 7E+61
    DURATION
        'Machine_Overflows    = FALSE
        'Machine_Rounds       = FALSE
        'Delta                = 2**(-14)
        'First                = -86400.0
        'Last                 = 86400.0
```

RESTRICTIONS ON MACHINE CODE INSERTIONS

Machine code insertions are not supported.